

```

1  package main
2
3  import (
4      "fmt"
5      "strconv"
6  )
7
8  type (
9      Node struct {
10         key   int
11         lson  *Node
12         rson  *Node
13     }
14     Tree struct {
15         root *Node
16     }
17 )
18
19 func InitTree() Tree {
20     return Tree{root: nil}
21 }
22
23 func (t *Tree) Insert(data int) {
24     t.root = t.insert(data, t.root)
25 }
26
27 func (t *Tree) insert(data int, node *Node) *Node {
28     if node == nil {
29         return &Node{key: data}
30     }
31     if data < node.key {
32         node.lson = t.insert(data, node.lson)
33     } else {
34         // data >= node.key {
35         node.rson = t.insert(data, node.rson)
36     }
37     return node
38 }
39
40 func (t Tree) Find(data int) bool {
41     return t.find(data, t.root)
42 }
43
44 func (t Tree) find(data int, node *Node) bool {
45     if node == nil {
46         return false
47     }
48     var result bool
49     switch {
50     case data == node.key:
51         result = true
52     case data < node.key:
53         result = t.find(data, node.lson)
54     case data > node.key:
55         result = t.find(data, node.rson)
56     }
57     return result
58 }
59
60 func (t Tree) Trace(up bool) []int {
61     // up - return tree in the ascending order
62     // !up - return tree in the descending order
63     return t.trace(t.root, up)
64 }
65
66 func (t Tree) trace(node *Node, up bool) []int {
67     if node == nil {
68         return []int{}
69     }
70     // node != nil
71     if up { // ascending order
72         return append(append(t.trace(node.lson, true), node.key),
73 t.trace(node.rson, true)...)
74     } else { // descending order
75         return append(append(t.trace(node.rson, false), node.key),
76 t.trace(node.lson, false)...)
77     }
78 }

```

```

79
80 func (t *Tree) Delete(data int) {
81     t.root = t.delete(data, t.root)
82 }
83
84 func (t *Tree) delete(data int, node *Node) *Node {
85     if node == nil {
86         return nil
87     }
88
89     if data < node.key {
90         node.lson = t.delete(data, node.lson)
91     } else if data > node.key {
92         node.rson = t.delete(data, node.rson)
93     } else // data == node.key
94         if node.lson != nil && node.rson != nil {
95             min := t.min(node.rson)
96             node.rson = t.delete(min.key, node.rson)
97             min.lson, min.rson = node.lson, node.rson
98             node = min
99         } else if node.lson == nil {
100             node = node.rson
101         } else { // node.rson == nil
102             node = node.lson
103         }
104     return node
105 }
106
107 func (t *Tree) min(node *Node) *Node {
108     for node.lson != nil {
109         node = node.lson
110     }
111     return node
112 }
113
114 func (tree Tree) TreeString() string {
115     return tree.treeString("", true, "", tree.root)
116 }
117
118 func (tree Tree) treeString(prefix string, top bool, str string, node *Node) string
119 {
120     if node == nil {
121         return ""
122     }
123     var temp string
124     if node.rson != nil {
125         if top {
126             temp = prefix + "|  "
127         } else {
128             temp = prefix + "  "
129         }
130         str = tree.treeString(temp, false, str, node.rson)
131     }
132     str += prefix
133     if top {
134         str += "└─"
135     } else {
136         str += "├─"
137     }
138     str += " " + strconv.Itoa(node.key) + "\n"
139     if node.lson != nil {
140         if top {
141             temp = prefix + "  "
142         } else {
143             temp = prefix + "|  "
144         }
145         str = tree.treeString(temp, true, str, node.lson)
146     }
147     return str
148 }
149
150 func (tree Tree) DraftDisplay() string {
151     return tree.draftDisplay("", tree.root)
152 }
153
154
155

```

```

156 func (t Tree) draftDisplay(prefix string, node *Node) string {
157     if node == nil {
158         return ""
159     }
160     return t.draftDisplay(" "+prefix, node.rson) +
161         prefix + strconv.Itoa(node.key) + "\n" +
162         t.draftDisplay(" "+prefix, node.lson)
163 }
164
165 func main() {
166     data := []int{34, 45, 36, 7, 24, 2, 40, 27, 5, 3}
167     tree := InitTree()
168     for _, key := range data {
169         tree.Insert(key)
170     }
171     fmt.Println(tree.DraftDisplay())
172     fmt.Println(tree.TreeString())
173     p := tree.root
174     fmt.Println(*p)
175     tree.Delete(34)
176     fmt.Println(*p)
177     fmt.Println(tree.Find(12)) // true
178     fmt.Println(tree.Find(18)) // false
179     fmt.Println(tree.Find(3)) // true
180     fmt.Println(tree.Trace(true)) // [2 3 5 7 7 12 15 16 24 27 34 36 40 45 52]
181     fmt.Println(tree.Trace(false)) // [52 45 40 36 34 27 24 16 15 12 7 7 5 3 2]
182     for _, key := range data {
183         if key%2 == 0 {
184             tree.Delete(key)
185         }
186     }
187     fmt.Println(tree.Trace(true)) //
188     fmt.Println(tree.Trace(false)) //
189     fmt.Println(tree.DraftDisplay())
190     fmt.Println(tree.TreeString())
191 }

```

bst.go

```

package main

import (
    "fmt"
    "strconv"
)

type (
    Node struct {
        key int
        lson *Node
        rson *Node
        height int
    }
    Tree struct {
        root *Node
    }
)

func InitTree() Tree {
    return Tree{root: nil}
}

func (t Tree) Trace(up bool) []int {
    // up - return tree in the ascending order
    // !up - return tree in the descending order
    return t.trace(t.root, up)
}

func (t Tree) trace(node *Node, up bool) []int {
    if node == nil {
        return []int{}
    }
    // node != nil
    if up { // ascending order
        return append ( append(t.trace(node.lson, true), node.key),
            t.trace(node.rson, true)...)
    } else { // descending order
        return append ( append(t.trace(node.rson, false), node.key),
            t.trace(node.lson, false)...)
    }
}

func (t *Tree) Find(key int) *Node {
    return t.find(key, t.root)
}

func (t *Tree) find(x int, node *Node) *Node {
    if node != nil {
        if x < node.key {
            return t.find(x, node.lson)
        } else if x > node.key {
            return t.find(x, node.rson)
        } else if x == node.key {
            return node
        }
    }
    return nil
}

//find min elem in the tree
func (t *Tree) MinNode() *Node {
    return t.minNode(t.root)
}

func (t *Tree) minNode(node *Node) *Node {
    if node != nil {
        if node.lson != nil {
            return t.minNode(node.lson)
        } else {
            return node
        }
    } else {
        return nil
    }
}

```

```

//insert an x to AvlTree
func (t *Tree) Insert(x int) {
    t.root = t.insert(t.root, x)
}

func (t *Tree) insert(node *Node, x int) *Node {
    if node == nil {
        //new(Node)
        node = &Node {
            key : x,
            height : 1,
        }
    } else
    if x < node.key {
        node.lson = t.insert(node.lson, x)
        if t.nodeHeight(node.lson) - t.nodeHeight(node.rson) == 2 {
            if x < node.lson.key { //left left
                node = t.singleRotateLeft(node)
            } else { //left right
                node = t.doubleRotateLeftRight(node)
            }
        }
    } else
    if x > node.key {
        node.rson = t.insert(node.rson, x)
        if t.nodeHeight(node.rson) - t.nodeHeight(node.lson) == 2 {
            if x >= node.rson.key {
                node = t.singleRotateRight(node)
            } else {
                node = t.doubleRotateRightLeft(node)
            }
        }
    }
    t.updateHeight(node)
    return node
}

//delete an x in AvlTree
func (t *Tree) Delete(x int) {
    t.root = t.delete(t.root, x)
}

func (t *Tree) delete(node *Node, x int) *Node {
    if node != nil {
        if x < node.key {
            node.lson = t.delete(node.lson, x)
            if t.nodeHeight(node.rson) - t.nodeHeight(node.lson) == 2 {
                if t.nodeHeight(node.rson.lson) <=
                    t.nodeHeight(node.rson.rson) {
                    node = t.singleRotateRight(node)
                } else {
                    node = t.doubleRotateRightLeft(node)
                }
            }
        } else
        if x > node.key {
            node.rson = t.delete(node.rson, x)
            if t.nodeHeight(node.lson) - t.nodeHeight(node.rson) == 2 {
                if t.nodeHeight(node.lson.rson) <=
                    t.nodeHeight(node.lson.lson) {
                    node = t.singleRotateLeft(node)
                } else {
                    node = t.doubleRotateLeftRight(node)
                }
            }
        } else {
    }
}

```

```

        // x == node.key
        if node.lson != nil && node.rson != nil {
            min:= t.minNode(node.rson)
            node.rson = t.delete(node.rson, min.key)
            min.lson, min.rson = node.lson, node.rson
            node = min
            if t.nodeHeight(node.lson) - t.nodeHeight(node.rson) == 2 {
                if t.nodeHeight(node.lson.rson) <=
                    t.nodeHeight(node.lson.lson) {
                    node = t.singleRotateLeft(node)
                } else {
                    node = t.doubleRotateLeftRight(node)
                }
            }
        } else {
            if node.lson == nil {
                node = node.rson
            } else
            if node.rson == nil {
                node = node.lson
            }
        }
    }
    t.updateHeight(node)
    return node
}

// left rotate a tree, and update node's height
// return the new root
func (t *Tree) singleRotateLeft(node *Node) *Node {
    var left *Node
    if node != nil {
        // turn left
        left = node.lson
        node.lson = left.rson
        left.rson = node

        //update height
        t.updateHeight(node)
        t.updateHeight(left)

        node = left
    }
    return node
}

// right rotate a tree, and update node's height
// return the new root
func (t *Tree) singleRotateRight(node *Node) *Node {
    var right *Node
    if node != nil {
        //turn right
        right = node.rson
        node.rson = right.lson
        right.lson = node

        //update height
        t.updateHeight(node)
        t.updateHeight(right)

        node = right
    }
    return node
}

```

```

// v = subtree root, vl = v's left child, vlr = vl's right child
// right rotate vl & vlr, left rotate v & v's left child
// return a new tree
func (t *Tree) doubleRotateLeftRight(v *Node) *Node {
    //right rotate1 between vl & vlr
    v.lson = t.singleRotateRight(v.lson)

    //left rotate between v and his left child
    return t.singleRotateLeft(v)
}

// v = subtree root, vr = vr's right child, vrl = vr's left child
// left rotate vr & vrl, right rotate v & v's right child
// return a new tree
func (t *Tree) doubleRotateRightLeft(v *Node) *Node {
    //left rotate1 between vr & vrl
    v.rson = t.singleRotateLeft(v.rson)

    //right rotate between v and his left child
    return t.singleRotateRight(v)
}

//return the height of the node
func (t *Tree) nodeHeight(node *Node) int {
    if node == nil {
        return 0
    } else {
        return node.height
    }
}

//recalculate the height of the node
func (t *Tree) updateHeight(node *Node) {
    if node == nil {
        return
    }
    if t.nodeHeight(node.lson) > t.nodeHeight(node.rson) {
        node.height = t.nodeHeight(node.lson) + 1
    } else {
        node.height = t.nodeHeight(node.rson) + 1
    }
}

func (t Tree) TreeString() string {
    return t.treeString("", true, "", t.root)
}

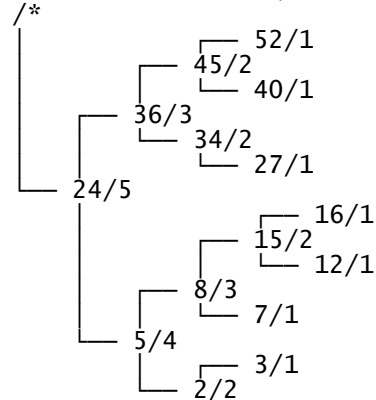
func (t Tree) treeString(prefix string, top bool, str string, node *Node) string {
    if (node == nil) {
        return ""
    }
    var temp string
    if (node.rson != nil) {
        if top {
            temp = prefix + "|  "
        } else {
            temp = prefix + "  "
        }
        str = t.treeString(temp, false, str, node.rson);
    }
    str += prefix
    if top {
        str += "└─"
    } else {
        str += "└─"
    }
    str += " " + strconv.Itoa(node.key) + "/" + strconv.Itoa(node.height) + "\n";
    if (node.lson != nil) {
        if top {
            temp = prefix + "  "
        } else {
            temp = prefix + "|  "
        }
        str = t.treeString(temp, true, str, node.lson);
    }
    return str
}

```

```

func main() {
    data:= []int{34, 45, 36, 8, 24, 2, 40, 27, 5, 3, 52, 7, 16, 12, 15}
    tree:= InitTree()
    for _, key := range data {
        tree.Insert(key)
    }
    fmt.Println(tree.TreeString())

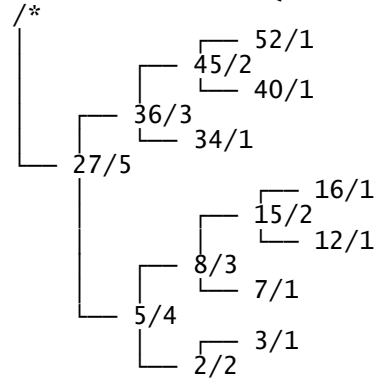
```



```

*/
    tree.Delete(24)
    fmt.Println(tree.TreeString())

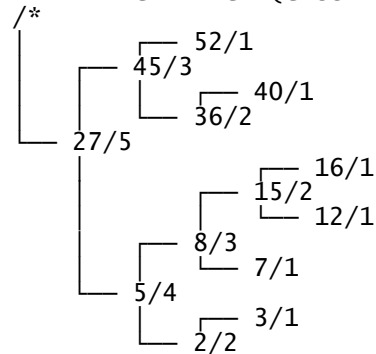
```



```

*/
    tree.Delete(34)
    fmt.Println(tree.TreeString())

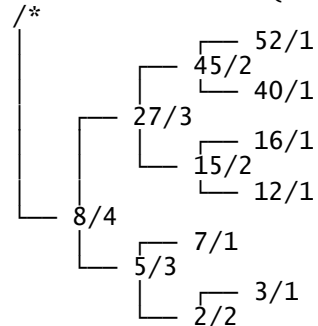
```



```

*/
    tree.Delete(36)
    fmt.Println(tree.TreeString())

```



```

*/
}

```